

Linear Algebra in Computer Graphics

Linear Algebra (MATH 206) Course Final Project

Authors: Audrey Ballarin, Caroline Jung, Heidi Salgado, Amy Hu, Rik Sampson

Table of Contents

1. Introduction to 2D Images and Image Editing with Filters and Shaders
2. Introducing the Problem: Image Compression
3. Discrete Cosine Transformation (DCT)
 - 3.1 The General DCT Process
 - 3.2 An Example
4. Singular Value Transformation (SVD)
5. Applications & Conclusion

1. Introduction to 2D Images and Image Editing with Filters and Shaders

Computer Graphics is the creation and manipulation of images in the form of pixels. In 1960, Verne Hudson and William Fetter from Boeing were the first to explore the perspective fundamentals of 3D modeling and computer animation of a human figure. Computer graphics have a variety of applications, from digital photography to film and video games. Anything that involves the usage of computers to create an image can be defined as computer graphics.

An RGB raster image can be represented as a matrix of pixels. Through applying transformation to the matrix, Linear algebra can be used to alter the image. This paper explores transformations such as shaders and image compression. The two topics in image compression that will be discussed are the Discrete Cosine Transformation (DCT) and Singular Value Decomposition which are lossy image compression methods.

An image of size $m \times n$ (e.g. 3400×4400 pixels) can be represented by a matrix of pixels. Most colored images use a RGB system for computer display, transmitted light, where each pixel is represented by an 8-bit binary number of the brightness level of RGB (red, green, blue) (Figure 1.1).¹ This binary number ranges from 00000000 to 11111111 where 0 represents black and 1 represents white. Numerically, the values range from 0 to 255. All the colors together create a 3D matrix made up of three layers of 2D matrices where each layer contains the R, G, or B values for each pixel. For grayscale pictures, which consist of a range of gray shades without color from black to white, the brightness levels of red, green, blue are all equal. Binary images, also referred to as “black and white” images or 1-bit pixel art, consist of pixels that only have two intensity values: either 0 (black) or 1 (white).² Binary images were especially common in displaying graphics in early computers.

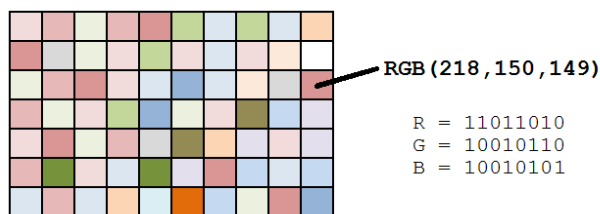


Figure 1.1: Representation of RGB values for a specified pixel. The red intensity level is 218, green intensity level is 150, and blue intensity level is 149. These numeric values are also represented in the corresponding binary numbers.³

¹ <https://www.techtarget.com/whatis/definition/grayscale>

² <https://towardsdatascience.com/a-visual-introduction-to-binary-image-processing-part-1-d2fba9f102a4>

³ <https://datagenetics.com/blog/march12012/index.html>

A fragment shader is a computer program that contains a set of instructions on how to render every single pixel. It returns the red, green, blue, alpha (i.e. transparency) which informs the color of each pixel. Many common filters can be applied through linear transformations represented by a 3x3 matrix. For example, grayscale conversion, sepia conversion, and individual color channel conversions (Figure 1.2). Just like regular vector transformations, the shader transformations can also be represented by unique matrices (Figure 1.3).



Figure 1.2: Original Image, Image after Grayscale Conversion, Image after Sepia Conversion (from left to right)

Grayscale Conversion:

$$T = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

Sepia Conversion:

$$T = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix}$$

Figure 1.3: Transformation Matrices for Grayscale Conversion and Sepia Conversion (from left to right)

A simple color change shader algorithm begins by obtaining the coordinate of each pixel in the shader and normalizing the coordinates to fit the display screen. For each pixel on the shader, the algorithm then maps it to the corresponding pixel on the image. Next, the algorithm alters the RGB-A value of the image pixel and sets the shader RGB-A values to the altered image RGB-A values for display. In the case of our example, the red value of each pixel is set to the value of the pixel's x-coordinate (Figure 1.4). Note that it is possible to alter the RGB-A value and/or map the image pixel to the shader pixel in any ways that the programmer would like. For example, it is possible to set the green value of each pixel to the value of the pixel's y-coordinate and/or map the entire image to a quarter of the shader on the screen.



Figure 1.4: Image before and after the color change shader algorithm (from left to right)

Shaders are useful not only for images, but for videos as well. One example of this is with green screens (aka chroma keying), a technique often used by the film industry, where scenes are filmed in front of a green background that is later digitally replaced by a different background. While the actual process for chroma keying is extremely complicated, shaders can be used to demonstrate a basic understanding of how to use a green screen. The basic ideas from using fragment shaders with images also applies to videos. The major distinction with green screen is that the difference between the RGB values of the original image and the defined RGB values of the green screen must first be calculated before altering the RGBA values of the video. If the difference is below a certain threshold, that indicates that the RGB values of the pixel are close to the green screen, and the pixel is replaced with the corresponding pixel from the background. Otherwise, the pixel values remain the same.

The three major operations commonly used in computer graphics are rotation, scaling, and translation. As linear transformations, rotation and scaling can both utilize their respective matrices. Rotating and scaling is as simple as multiplying the vector representing the pixel position with the matrix of the desired transformation. However, the number of pixels in an image is often very large, and computing rotations for thousands of pixels by hand is not realistic. Fragment shaders help with this, almost instantaneously calculating the transformation for each pixel, though this will only result in a static image where the initial pixels are rotated or scaled once.

To achieve an animated effect with a shader, a uniform time variable must be used. This is essentially the amount of time in seconds that has elapsed since loading. Imagine an image with a square in the middle. Using uniform time as the angle in a rotation matrix results in a continuous rotation of that square. In the case of scaling, doing this would result in the infinite

scaling of the square, and the scale would soon be out of view. As a variable, uniform time can be used with a variety of different operations. For instance, scaling the square by the sine of uniform time results in the square alternating between scaling up and down in size.

In computer graphics, it is common for multiple operations to be performed at once. While optimization of the program isn't that important for simple examples in 2D graphics, it is a major concern for high-end graphics. Matrix composition is a convenient solution to this. Instead of applying individual transformations to the vector one at a time, it is much easier to multiply all the transformation matrices together to form one matrix, which is then multiplied with the vector. Rotating the square while simultaneously scaling it is as simple as multiplying the rotation and scale matrices together. However, a problem arises when translating the square while also rotating and scaling it. Translation uses addition, not the matrix-vector multiplication utilized by the rotation and scale transformations. Because of this, translation as it is originally cannot be part of the matrix composition.

Homogeneous coordinates address this problem by increasing the dimension of the coordinates. In the case of translation in 2D graphics, the translation goes from being a vector into a 3x3 matrix that can be used in matrix composition. Rotation and scale transformations must also be expressed in homogeneous coordinates, turning them into 3x3 matrices as well, but it is now possible to use one matrix to simultaneously translate, rotate, and scale the square, instead of applying each transformation individually.

There exists many 2D image processing examples that utilize Linear Algebra. One example is the Affine Transformations. Affine Transformations deals with linear transformation that preserves linearity so if points are lying in a line, then after the transformation, they will remain on a line. Generally, affine transformations compose of translations, rotations, scalings and many other transformations. In connection with Computer Graphics, affine transformations are utilized to alter the position of the pixels inside the image and not the pixels themselves like shaders and filters. Another 2D image processing example is Image Restoration. Image Restoration is the process where you have a blurry image or an image with variations of color and brightness, then improving the quality to get a clean image through transformations and filters. Common types of restoration filters consist of inverse filters and Wiener filters. We have

seen before how Linear Algebra is utilized in filters, which can also apply to the restoration filters used in the Image Restoration.

2. Introducing the Problem: Image Compression

Narrowing down a problem, this paper will focus on the topic of image compression. Image compression is a form of data compression that reduces the amount of storage an image requires (Figure 2.1).



Figure 2.1: An example of a high quality image (left) compressed to a lower quality image (right)⁴

Image compression is used to reduce storage space. It reduces file size by removing or reworking data and makes it more efficient to retrieve, transfer, and upload photos. There are two main types of data compression that image compression also uses: lossy and lossless.⁵

Lossy compression is frequently used when there is very limited storage space or images need to be frequently transferred between platforms. It permanently removes unnecessary data, data that is not detected by the human eye, through algorithms (two of which include Discrete Cosine Transform and Singular Value Decomposition). Lossy compression is frequently supported by other tools, plugins, and software, but it decreases the image quality. An example of lossy compression is JPEG.

On the other hand, lossless compression also reduces file size, but not to the extent of lossy compression and it does not usually impact image quality. It removes unnecessary metadata which can be restored or rebuilt, commonly called “reversible compression”. Examples of lossless compression include PNG, RAW, and BMP.

⁴ <https://www.rothgalleries.com/gallery-image/Massachusetts/G0000uFrJYrQlk.M/I0000cHlah4XGfmE>

⁵ <https://www.adobe.com/uk/creativecloud/photography/discover/lossy-vs-lossless.html>

The following sections (Sections 3 and 4) cover two techniques for lossy image compression: Discrete Cosine Transformation (DCT) and Singular Value Decomposition (SVD).

These two techniques must be mentioned when mentioning the history of image compression. In 1873-4, SVD, a form of matrix decomposition, was independently established by five mathematicians: Eugenio Beltrami, Camille Jordan, James Sylvester, Erhard Schmidt, and Hermann Weyl.⁶ In 1973, DCT was proposed by three mathematicians: N. Ahmed, T. Natarajan, K.R. Rao.⁷ In 1992, the JPEG compression standard was introduced which uses the DCT algorithm.⁸ JPEG compresses images down to smaller file sizes, a reason why it is responsible for the wide proliferation of digital images.

3. Discrete Cosine Transformation (DCT)

3.1 The General DCT Process

The Discrete Cosine Transformation, which will be referred to as DCT, is a form of lossy data compression that was originally intended for images. It is used by JPEG compression standards where it removes as much data as possible while preserving the general image. It transforms signal data, the image, into spectral or frequency data, numeric data, which is represented by sinusoids of varying magnitudes and frequencies.⁹ This data is then stored as a JPEG which is then read by the computer to develop the compressed image. Since it deals with frequencies, the DCT is a Fourier-related transformation but with real numbers. The general process for DCT starts off with separating the original image into parts, determining the important information or shades about the image, and removing unnecessary data to get the final compressed image.¹⁰

⁶ <https://epubs.siam.org/doi/10.1137/1035134#:~:text=The%20singular%20value%20decomposition%20>

⁷

<https://www.scribd.com/doc/52879771/DCT-History-How-I-Came-Up-with-the-Discrete-Cosine-Transform>
⁸ <https://www.theatlantic.com/technology/archive/2013/09/what-is-a-jpeg-the-invisible-object-you-see-every-day/279954/>

⁹ <https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/jpeg/dct.htm>

¹⁰

[https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html#:~:text=The%20discrete%20cosine%20transform%20\(DCT\)%20helps%20separate%20the%20image%20into.frequency%20domain%20\(Fig%207.8\).](https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html#:~:text=The%20discrete%20cosine%20transform%20(DCT)%20helps%20separate%20the%20image%20into.frequency%20domain%20(Fig%207.8).)

The first step in DCT is to separate an $m \times n$ image into 8×8 blocks of pixels to formulate a matrix of 64 elements. The original image is represented by matrix A where each block of pixels is represented by A_{mn} (Figure 3.1).

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots \\ A_{21} & \ddots & \\ \vdots & & A_{88} \end{bmatrix}$$

Figure 3.1: Matrix A represents the original image split up into an 8×8 matrix where each block of pixels in the image is represented by A_{mn} .

The second step is to perform the DCT algorithm for each A_{mn} to result in the DCT coefficient matrix. Each A_{mn} undergoes the DCT transformation (Equation 3.1) to result in transformed values called coefficients which are denoted as $B_{p,q}[m,n]$.¹¹ These coefficients make up a matrix called the DCT coefficient matrix which is represented by matrix B (Figure 3.2). The DCT coefficient matrix has a basis function where the coefficients $B_{p,q}[m,n]$ can be thought of as the weights for each basis function (Figure 3.2).

$$B_{p,q}[m,n] = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\left(\frac{\pi(2m+1)p}{2M}\right) \cos\left(\frac{\pi(2n+1)q}{2N}\right) \quad \begin{matrix} 0 \leq p \leq M-1 \\ 0 \leq q \leq N-1 \end{matrix}$$

\uparrow Indices of frequencies \uparrow Indices of image A

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}} & \text{if } p = 0 \\ \frac{1}{\sqrt{\frac{2}{M}}} & \text{if } 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } q = 0 \\ \frac{1}{\sqrt{\frac{2}{N}}} & \text{if } 1 \leq q \leq N-1 \end{cases}$$

Equation 3.1: The DCT transformation algorithm. For each A_{mn} , the corresponding coefficient value, $B_{p,q}[m,n]$, can be calculated. M and N represent the size of the image ($m \times n$) and p and q represent the vertical and horizontal cosine frequency, respectively.

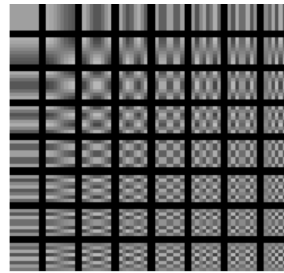
$$B = \begin{bmatrix} B_{0,0}[m,n] & B_{0,1}[m,n] & \cdots \\ B_{1,0}[m,n] & \ddots & \\ \vdots & & B_{7,7}[m,n] \end{bmatrix}$$


Figure 3.2: (Left) The DCT coefficient matrix, denoted as matrix B , is made up of coefficients $B_{p,q}[m,n]$. (Right) The 64 Basis Functions of an 8×8 matrix for DCT, visually represented.

The third step is to save the important and significant coefficients. This is done by analyzing the DCT coefficient matrix, matrix B . The significant coefficients are concentrated in the upper left corner (orange shaded area) where there are low frequencies p and q as well as

¹¹ <https://www.mathworks.com/help/images/discrete-cosine-transform.html>

large magnitudes. The non-significant coefficients are concentrated in the lower bottom corner (pink shaded area) with high frequencies p and q and small magnitudes (Figure 3.2). Since the human eye is not sensitive to high frequencies, the coefficients with high frequencies are considered to be not significant.¹² Thus, only the low frequency coefficients (orange area), representing shades or colors, are chosen to be used for image compression.

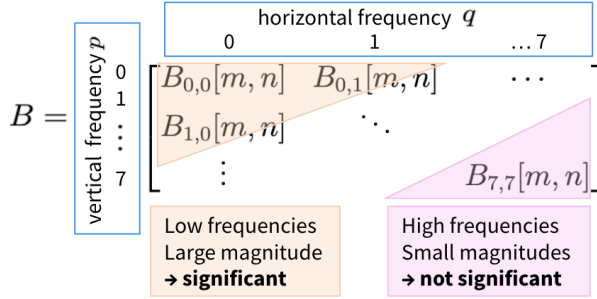


Figure 3.2: The DCT coefficient matrix labeled with significant vs non-significant coefficients. Significant coefficients are concentrated in the orange region and non-significant coefficients are concentrated in the pink region.

The fourth step in the DCT process is to allocate bits based on each coefficient's significance. The more bits a coefficient has, the more significant it is deemed. This information on bits is stored in a JPEG file.¹³ At this point, the important information about the original image is stored in a file, but to actually obtain the compressed image, the JPEG must be decoded.

Thus, the fifth step is to read the number of bits each coefficient has from the JPEG file. The computer would be able to convert the number of bits to the coefficients $B_{p,q}[m,n]$ themselves.

The sixth step is to perform inverse DCT on each of these coefficients to find elements A_{mn} of matrix A . Recall that the second step utilized the DCT algorithm to transition from A_{mn} to $B_{p,q}[m,n]$. Therefore, to find A_{mn} , inverse DCT must be used (Equation 3.2). Note that A_{mn} represents blocks of pixels of the image.

$$A_{mn} = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \alpha_p \alpha_q B_{p,q}[m,n] \cos\left(\frac{\pi(2m+1)p}{2M}\right) \cos\left(\frac{\pi(2n+1)q}{2N}\right)$$

¹² <https://www.youtube.com/watch?v=DS8N8cFVd-E>

¹³ <https://www.youtube.com/watch?v=DS8N8cFVd-E>

$$\alpha_q = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } q = 0 \\ \sqrt{\frac{2}{N}} & \text{if } 1 \leq q \leq N - 1 \end{cases} \quad \alpha_p = \begin{cases} \frac{1}{\sqrt{M}} & \text{if } p = 0 \\ \sqrt{\frac{2}{M}} & \text{if } 1 \leq p \leq M - 1 \end{cases} \quad \begin{matrix} 0 \leq m \leq M - 1 \\ 0 \leq n \leq N - 1 \end{matrix}$$

Equation 3.2: The inverse DCT algorithm. For each $B_{p,q}[m,n]$, the corresponding A_{mn} can be calculated. M and N represent the size of the image ($m \times n$) and p and q represent the vertical and horizontal cosine frequency, respectively.

After obtaining each A_{mn} value, these elements are combined together to develop the final compressed image. This is the seventh and last step of this process.

3.2 An Example

This DCT process can be exemplified by an example. From an original image, the first step is to split up the image into an 8x8 matrix (Figure 3.3). Each square in the matrix, denoted by A_{mn} , represents a block of pixels from the original image. Note that in this specific example, the image is grayscale, meaning that it is made up of shades of gray instead of colors.

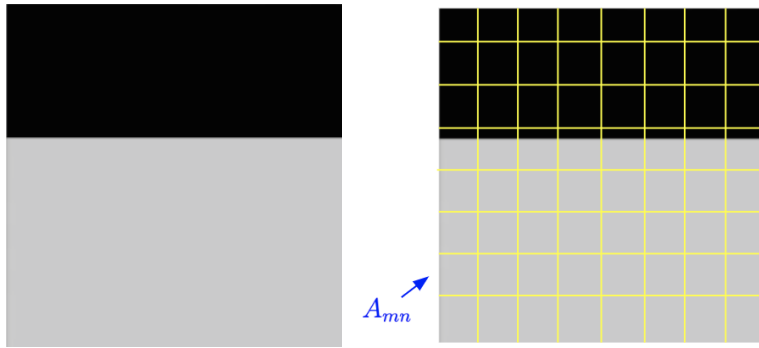


Figure 3.3: (Left) An example of an original image. (Right) Splitting the original image into an 8x8 matrix.

The DCT transformation is then performed for each block of pixels A_{mn} which returns coefficient values $B_{p,q}[m,n]$ that make up the DCT coefficient matrix. The upper left coefficients, also referred to as shades in this example, are significant. In this example, four shades are chosen, as indicated by the yellow tabs since those shades are more distinguishable than shades in the lower right of the matrix (Figure 3.4). Note that all of our shades appear where the index of the horizontal frequency q (column headings) is 0. This is consistent with the original image since there is no fluctuation in the shades when observing the image horizontally (Figure 3.3).

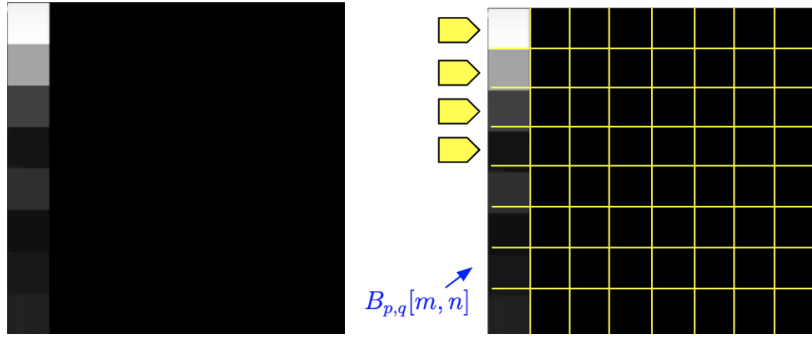


Figure 3.4: (Left) The DCT coefficient matrix. (Right) Diagram of the coefficient matrix labeled with the specific four coefficients, or shades, chosen.

After determining the significance of the coefficients, the number of bits are allocated and stored in a file. When read by the computer, it converts the number of bits into coefficients $B_{p,q}[m,n]$, the individual shades to use in the compressed image. These coefficients are then transformed via inverse DCT into elements A_{mn} of matrix A , the blocks of pixels in the compressed image. Once these elements are combined together, the final compressed image is developed.



Figure 3.5: The final compressed image from the original image.

As a quick summary, the original image is split up into an 8 x 8 matrix A with elements A_{mn} which are then transformed into coefficients $B_{p,q}[m,n]$ through DCT. These coefficients make up the DCT coefficient matrix that reveals the significant coefficients. Only the significant coefficients are chosen to be used in the compressed image. This process is visually represented below (Figure 3.6).



Figure 3.6: (Left) The original image. (Middle) The DCT coefficient matrix. (Right) The compressed image.

4. Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is another methodology used in image compression, as well as in other forms of data compression. SVD entails the decomposition of an input matrix into three matrices, U , Σ , and V^T . Given an m -by- n input matrix A that is decomposed to $U\Sigma V^T$, U is an m -by- k orthogonal matrix of column vectors containing the *left singular values*. V^T is a k -by- n orthogonal matrix of row vectors containing the *right singular values*. Σ is a k -by- k diagonal matrix where the diagonal values contain information about the weights of the left and right singular values. The sigma values are square roots of the *eigenvalues* of the matrix.

In terms of graphics, a grayscale image is represented by a matrix where each value represents a pixel in a color between 0 and 1, where black is 0 and white is 1. Lossy image compression entails the selection of the most “important” (typically the most frequently used) color values, and the elimination of the less important values. SVD relates to image decomposition in that the vectors of U and V^T are hierarchically arranged so that the more important color values are placed first. The hierarchical spatial arrangement of color values in these matrices reflects a similar concept to the spatial arrangement of important values to the top left of a DCT matrix. The sigma values across the diagonal eventually become zeros, so when they are used as weights for the left and right singular values in the matrix multiplication, they effectively cancel out the “less” important pixels, resulting in a simpler, compressed version of the original matrix.

5. Applications & Conclusion

Through our discussions on Discrete Cosine Transformation (DCT) and Singular Value Decomposition (SVD), linear algebra played a key role in the process of image compression. The linear operations helped in improving speed and accuracy when compressing images. As discussed before, image compression has many benefits especially when it comes to computer graphics technology such as films, digital photography, web-browsing and many more. A few key benefits consists of the following: saving storage space, faster uploads and faster website loads. Therefore, there exists multiple compression methods that are widely used such as the JPEG compression tool, which has been around for quite some time now. However, as technology advances, there is always a need for faster and more efficient methods. Therefore, new types of file compression methods are rising such as HEIF, Guetzli and Mozjpeg. For instance, the High-Efficiency Image Format, also known as HEIF, tool is shown to compress images at about half the size of JPEG images with better quality.¹⁴ Whereas, Guetzli is a JPEG compression that offers smaller image sizes by 20%-30%.¹⁵ Many popular companies are adopting these methods to beat others in efficiency and speed. For example, Apple is using HEIF on iOS 11 and macOS High Sierra models and above, and Google is using Guetzli which would be more supported across the web due to the standardization of JPEG compression.¹⁶ Overall, with new compression tools on the rise, the use of linear algebra is vital in helping to improve or introduce new methods and algorithms.

In conclusion, linear algebra plays a vital role in the concepts of computer graphics. We only covered a fraction of 2D image processing problems that use linear algebra for a solution. As technology advances, the concept of linear algebra will be fundamental in understanding Computer Graphics.

¹⁴ <https://www.wtmdigital.com/blog/future-image-compression/>

¹⁵ <https://www.wtmdigital.com/blog/future-image-compression/>

¹⁶ <https://www.wtmdigital.com/blog/future-image-compression/>